CS 342 Design Document
Project 2B and 2C : User Memory Access and Exit and Write System Calls

Shrinivas Acharya
10010164

---- DATA STRUCTURES ----

No new data structures were added for this project as all it required was memory access checking and implementing system calls

---- ALGORITHMS ----

Checking user memory access

For checking that the user program is accessing only that portion of the memory which it is authorised to use, the inbuilt function lookup_page has been used. The function is defined in the ./src/userprog directory. It takes three arguments, first is the pagedirectory associated with each thread, then the logical address which is to be checked, and last is a boolean variable. Providing the boolean variable as false, if the address to be checked is not valid, the function returns a NULL pointer. So the return value is checked, and if it is NULL, the thread exits with exit code -1 representing an error exit.

The same is also done with all address (pointers) which are received as arguments to certain system calls, for example in SYS_WRITE system call, a char pointer is received. It is also checked for validity, and only when the pointers are valid, the function executes, otherwise it exits with error exit status -1.

SYS_EXIT system call

The system exit call is called with an exit status as argument. Firstly that argument is read from the stack (the address used is of stack pointer + 4 because each argument takes up 4 bytes). Then that argument is written onto the eax register which stores the return value. Then the exit status is printed as required by the implementation. Finally thread_exit() is called to kill the thread.

SYS_WRITE system call

The write system call receives three arguments, the file no to write onto, the char pointer buffer from where to write and the size of data to write. For now, only the case of file number 1 is implemented, which is writing on the standard output. For that, the inbuilt function putbuf is used. It writes the size bytes of data from the buffer onto the standard output.

---- SYNCHRONIZATION ----

For this project there were no race conditions, hence no particular synchronisation was required.

---- RATIONALE ----

The design used for implementing the system calls, as well as user memory access checking, make use of the inbuilt functions, which provide the advantage that they consider all the boundary cases themselves, and hence no extra code was required to handle the boundary cases.

------ CODE SNIPPETS ------

User memory access

```
struct thread *cur = thread_current();
uint32_t *valid = lookup_page(cur->pagedir, f->esp, false);
if(valid == NULL)
{
	f->eax = -1;
	printf("%s: exit(%d)\n", cur->name, -1);
	thread_exit();
}
```

SYS_EXIT system call

```
case(SYS_EXIT):
{
	int status = * (int *) (f->esp + 4);
	f->eax = status;
	printf("%s: exit(%d)\n", cur->name, status);
	thread_exit();
	break;
}
```

SYS_WRITE system call

```
case(SYS_WRITE):
{
	int file_no = * (int *) (f->esp + 4);
	char *buffer = * (char **) (f->esp + 8);
	if(lookup_page(cur->pagedir, buffer, false) == NULL)
	{
		f->eax = -1;
		printf("%s: exit(%d)\n", cur->name, -1);
		thread_exit();
	}
	int size = * (int *) (f->esp + 12);
	if(file_no == 1)
	{
		putbuf(buffer, size);
	}
		break;
}
```